Advanced Systems Security: Capability Systems

Trent Jaeger
Systems and Internet Infrastructure Security (SIIS) Lab
Computer Science and Engineering Department
Pennsylvania State University

Confused Deputy

PENNSTATE 1855

- Is there another approach to preventing confused deputy attacks?
- Yes, it is called a capability system

Overview of Solution

PENNSTATE 1855

- Server accepts client requests
  - Which include a reference to the object that the client wants to operate on
  - The reference identifies the object and includes the client's permissions
- Server only uses client capabilities to perform client requests
  - Server uses its own permissions for its internal operations
  - Server must not confuse its own capabilities and its clients' capabilities, but that is easier than filtering, etc.

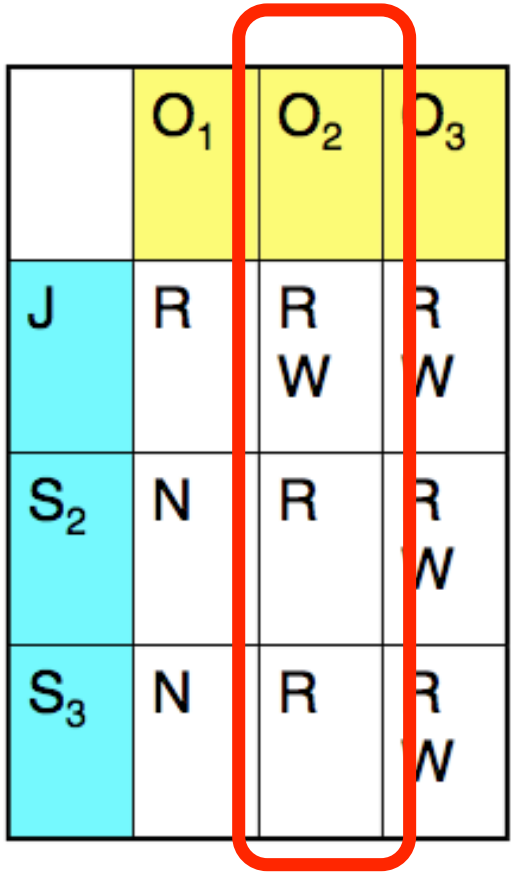
Access Matrix

- Back to the access matrix

	O_1	O_2	O_3
J	R	R W	R W
S_2	N	R	R W
S_3	N	R	R W

Access Matrix

- Access Control Lists: Ordinary systems use those



	O_1	O_2	O_3
J	R	R W	R W
S_2	N	R	R W
S_3	N	R	R W

Access Matrix

- Capability Lists: An alternative representation of the same thing, but…

	O_1	O_2	O_3
J	R	R W	R W
S_2	N	R	R W
S_3	N	R	R W

Capability-Based Addressing

PENNSTATE 1855

- Goes back to the mid-1960s (Dennis and van Horn, Plessey system, CTSS)
- Idea: include accessibility with reference
- What is a normal reference?
- What defines accessibility?

Capabilities

- Analogy
- Like a house key
  - Possession grants access
  - Need to use the right key for the right job
  - Can make copies and give those to others
  - Changing the lock invalidates all keys
  - Losing the key loses access
  - Can't easily keep track of where the copied keys go

What's a Capability?

PENNSTATE 1855

- Consists of a reference
  - Object ID, memory value, segment number, label, …
- And rights
  - Operations specific to that object type (class in SELinux)
- And an integrity value (optional)
  - Needed if a capability may be handled by an untrusted party (like communicating a message securely)
- Present this to an object server to obtain access to the reference to use the rights

Capability Requirement

PENNSTATE 1855

- Capabilities must be unforgeable
  - Why would a user forge a capability?
- Under what conditions should we worry about forgery?

Capability Requirement

- Capabilities must be unforgeable
  - Why would a user forge a capability?
- Under what conditions should we worry about forgery?
  - Users hold their own capabilities
  - Users convey capabilities across untrusted channels

Capability Requirement

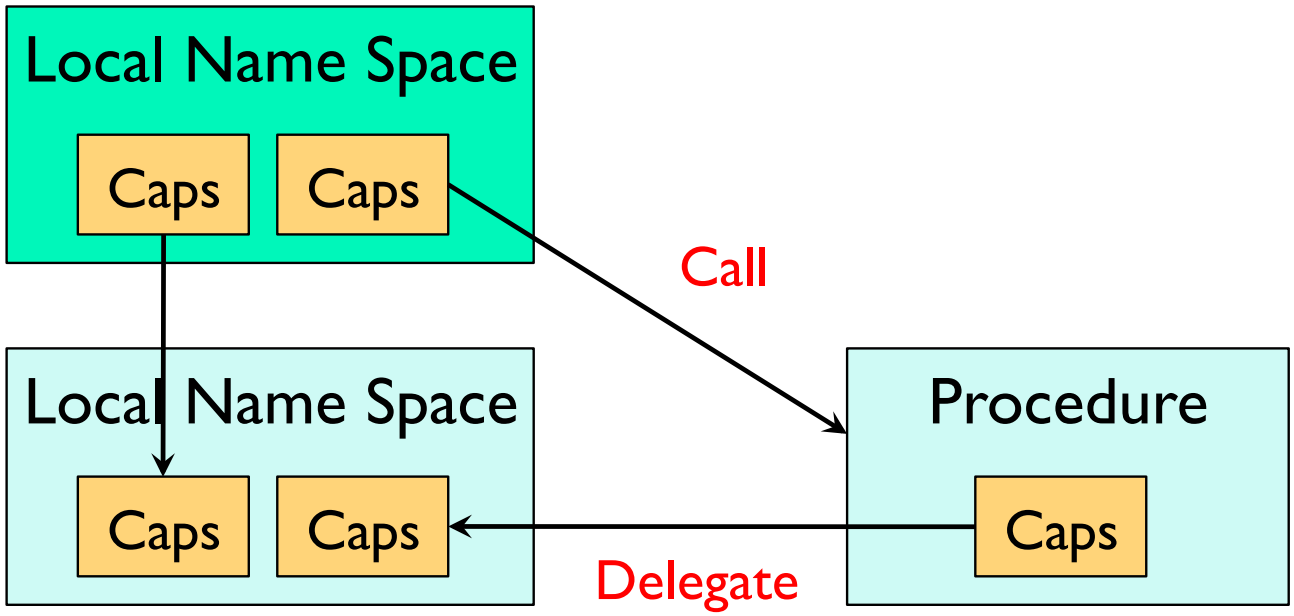
PENNSTATE 1855

- Capabilities must be unforgeable
  - Why would a user forge a capability?
- Representations of Capabilities
  - Hardware capabilities
    - Hardware associates permissions with reference
  - System-controlled capabilities
    - System stores mapping of permissions to reference
  - Cryptographic capabilities
    - User processes hold and distribute capability objects

Hydra System

- "Everything is an object" capability system
  - Where objects and code may be associated with capabilities to access those
- Access control
  - C-List: each process has capabilities to access objects
- Processes are objects, as are procedures
  - Protection at procedure granularity
- *Your rights are based on the procedure you are currently executing*

Hydra System



All authorized operations of a procedure are defined by its (inherited) capabilities and those passed by the caller

Capability Confinement Problem

- Boebert: *"the right to exercise access includes the right to grant access"*
  - Why is that a problem?

Capability and *-Property

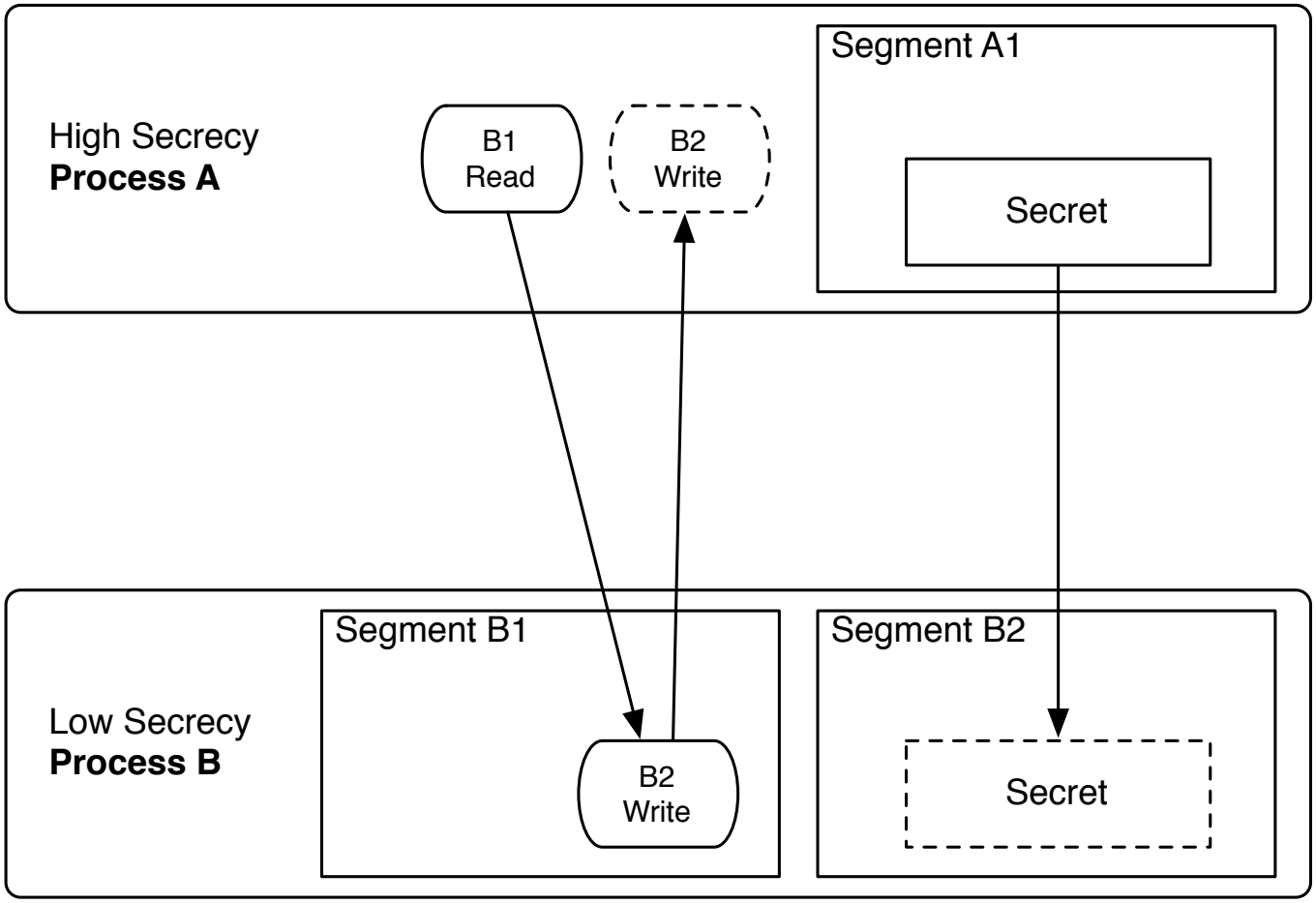


Figure 10.1: A problem with the enforcing the ⋆-property in capability systems

Capability Confinement Problem

- Boebert: *"the right to exercise access includes the right to grant access"*

- If I can talk to you, I can give you permissions
  - Low process can give high process a capability to leak secret data (*-property violated)
  - And leak other capabilities to objects the low process can be read to further exploit access (no confinement)
  - And no mechanism to get these capabilities back (need revocation)

Difference from Access Matrix

- Capability-Based Addressing: Does not include identity for authorization system to check

	O_1	O_2	O_3
J	R	R W	R W
S_2	N	R	R W
S_3	N	R	R W

Anyone can use – regardless of the access matrix

Protection vs. Security

PENNSTATE 1855

- Consider a benign process
  - If it has a fault, will it leak a capability?
  - Will it receive another's capability to leak information?
  - Will it forge a capability?
- Consider a malicious process
  - It will try to leak a capability
  - It will try to leak information
  - It will try to forge a capability
- Capability systems aim for protection, not security

What to do?

Security Issue	SCAP Solution	EROS Solution
$\star$-Property	Convert to *read-only* capabilities by MLS policy	Define *weak* capabilities that transitively fetch only read-only capabilities
Confinement	Use Access Control List to define confinement	Define *safe* environments for confined processes or test via *authorize* capabilities
Revocation	Revocation by eventcounts (single page entry) or revocation by chaining (multiple page entries)	Indirect capabilities that permit later revocation of all descendants (similar to Redell [251])

Table 10.1: Summary of SCAP and EROS solutions to the major security issues in capability systems.

SCAP and EROS

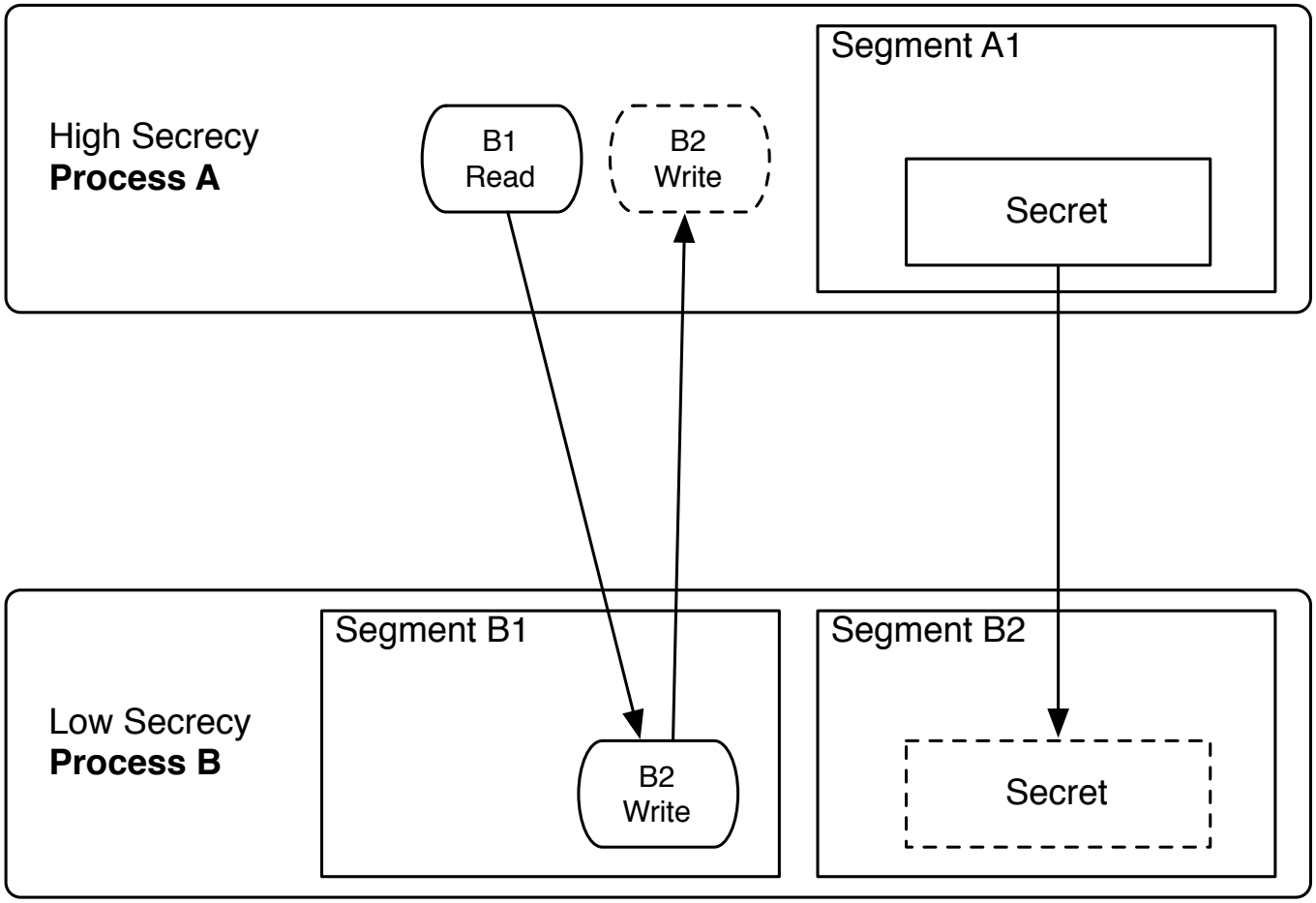


Figure 10.1: A problem with the enforcing the $\star$-property in capability systems

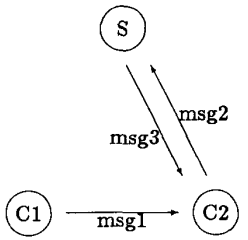
EROS *-Property

PENNSTATE 1855

- Confinement limits access, so that a high secrecy subject cannot use a write-capability to a low secrecy object
  - Validate for yourself
- EROS – use a *weak capability*
  - Give a high secrecy process a *weak capability* to read from a low secrecy object
  - Any capabilities obtained via this capability are made *read-only* and *weak*
  - Couldn't a Trojan horse still read memory and then provide that as a capability later?

Confinement

- Restrict permissions to satisfy a policy for the presenting subject
  - Rather than simply permitting access via possession
- SCAP
  - Need cap and policy to authorize
- EROS
  - Test capability set in advance
  - Or authorize via policy



1. A classic capability system.

   msg1 : C1 propagates a capability cap1 to C2.

   msg2 : C2 requests the access by presenting cap1 to S.

   msg3 : S checks the validity of cap1 and grants access.

2. Karger's SCAP.

   msg1 : C1 propagates cap1 to C2.

   msg2 : C2 requests the access by presenting cap1 to S.

   msg3 : S checks both cap1's validity and whether the access complies with the security policy. *If so it grants the access.*

ICAP

- Add the identity of the holder into the capability
  - (client ID, obj ID, rights, integrity)
- Authorize transfer of capabilities
  - C1 constructs a signed message to grant a capability to C2 – C1 must be the identity in the capability
  - C2 presents capability and signed grant on first use
  - Server authorizes based on security policy and creates a capability for C2 to use

Revocation Problem

PENNSTATE 1855

- How do I get the house key I copied for you?
  - Without changing my locks…
- It is not practical to scan through memory to find capabilities

Revocation – Redell's Solution

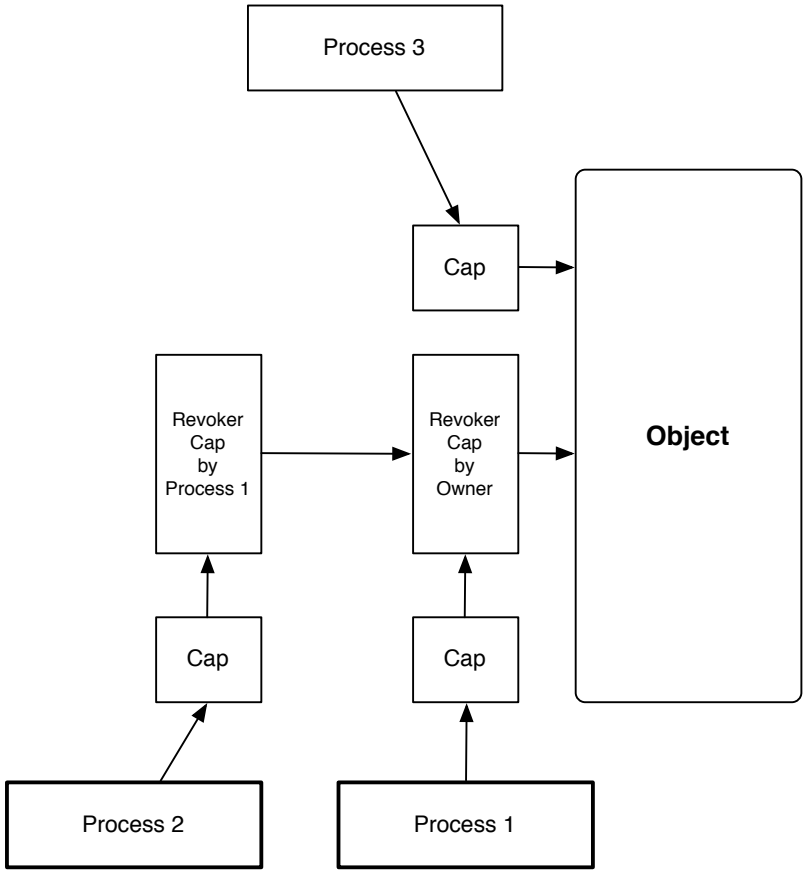
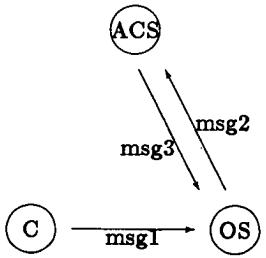


Figure 10.2: Redell's revoker capability approach: When the revoker capability is revoked all the capabilities that were based on it are also revoked.

Revocation – Others

- Other solutions are a bit convoluted
- SCAP
  - Event counts: compare page table entry count to capability
    - Revalidate if different
- ICAP
  - Revocation list with Redell's propagation tree

Overall: ICAP



Creation
1. C requests to create a new object (msg1).
2. OS creates the object together with a new internal capability and reports this to ACS (msg2). It also creates and returns an external capability to C.
3. ACS updates the access control list. It can create a root for the propagation tree of this object.

Access
1. C presents the external capability (msg1). OS retrieves the corresponding internal capability and runs the one-way function to do a validity check and decide whether to grant access.

Propagation
1. C requests a propagation (msg1).
2. OS asks ACS whether the request is allowed by the policy (msg2).
3. ACS checks the policy and replies accordingly (msg3). If it is allowed, ACS records the propagation in the corresponding tree.
4. Upon getting a positive reply OS creates and returns a new external capability.

Revocation
1. C requests a revocation (msg1).
2. OS updates the exception list pending the access temporarily. Then it consults ACS as to whether the revocation is legal (msg2).
3. ACS checks the propagation tree and replies (msg3). If it is legal, ACS decides whether to arrange for a full revocation.
4. Upon receiving msg3, if the revocation is legal, OS marks it permanent and notifies C if necessary; otherwise, it resolves the pending access.
5. When a full revocation is done, ACS notifies OS. OS replaces the old capability in its internal table and deletes the entry from the exception list.

CHERI

PENNSTATE 1855

- Capability system to make capability use seamless and efficient
- To sandbox code within a process
  - Untrusted code to run in your address space
  - Without allowing unauthorized access to modify and/or read other, sensitive process data
- Challenges
  - Make capabilities unforgeable
  - Without appealing to the kernel

CHERI Overview

PENNSTATE 1855

- Capability system to make capability use seamless and efficient
  - Hybrid capability model for intra-address space
- Key features
  - Capability coprocessor that provides capability registers, similar to segment descriptors (see Multics)
  - Tagged memory to distinguish in-memory capabilities from regular memory (common approach from past)
- Use capabilities to check bounds, control access, and protect pointer integrity within address space

CHERI Requirements

PENNSTATE 1855

- Requirements for intra-address space protection
- Access control – for memory regions
- Unforgeability – no privilege escalation
- Fine-grained – support small and dense regions
- Unprivileged use – no system call required
- Overhead – scale with number of memory regions, number of domains, and intercommunications
- Legacy – work with recompilation

CHERI Memory Model

PENNSTATE 1855

- Memory capability model
  - A memory capability is an unforgeable pointer that grants access to a linear range of the address space
  - All memory accesses must occur through memory capabilities
    - What about legacy code and its pointers?
- Protection domain of a process
  - Is the transitive closure of the capabilities reachable from its own capabilities

CHERI Capabilities

PENNSTATE 1855

- Capabilities are 256-bit "fat" pointers
  - Base memory address and length (memory segment)
  - Permissions (access control in 31 bits)
- Protected by tagged memory
  - User-mode instructions can load/store caps and reduce privileges
  - Process starts with capability to full address space and creates more restricted capabilities for other domains
- Enables legacy code to launch capability-aware code and vice versa

CHERI Use Cases

PENNSTATE 1855

- Memory bounds enforcement
  - As "fat" pointers
    - Base and length
    - Natural for the heap
    - Can also be applied to the stack – bit more ad hoc
- Sandboxing
  - Create "micro" address spaces by constraining code and data capabilities
- Limit - Need to modify and recompile source code

CHERI Domains

PENNSTATE 1855

- Protection domain of a process
  - Is the transitive closure of the capabilities reachable from its own capabilities
- An issue for Boebert's claim?

Take Away

- Problem: Control Access and Confused Deputy
  - Using a "key" is a natural way to control access
    - No centralized service required
  - Prevent need for a server to manage all its clients permissions
- Unfortunately, neither of these problems can be completely solved by capabilities
  - Confinement: Need identity to control access – end up with a centralized access server (holds or verifies perms)
  - Revocation: Need to track delegation somewhat